基于熵编码的 JPEG 压缩域脆弱图像水印算法 *

李　晨，喻　枭，田丽华†

(西安交通大学 软件学院, 西安 710049)

摘　要：为了解决 JPEG 图像水印算法在嵌入容量、不可感知性、实时性和窜改定位的问题，提出了一种基于熵编码的 JPEG 压缩域脆弱图像水印算法。首先选择 JPEG 熵编码阶段进行水印嵌入，这样就可避免正向和逆向的 DCT 和量化运算，保证算法在嵌入阶段对图像的改变量较小且具有较好的实时性；接着根据熵编码过程中某些比特不会参与哈夫曼编码这一原则，在其中选择符合条件的系数嵌入水印，将水印信息嵌入到这些系数的最低有效位中，以规避修改哈夫曼编码系数后带来的编码误差，进一步加强了算法的透明性。实验表明，该算法不仅具有较大的嵌入容量和很好的不可感知性，而且脆弱性较高且能够对恶意窜改进行准确定位。

关键词：图像水印；脆弱水印；JPEG 图像；窜改定位

中图分类号：TP309.2　　doi: 10.3969/j.issn.1001-3695.2018.01.0098

JPEG compressed domain fragile image watermarking algorithm based on entropy coding

Li Chen, Yu Xiao, Tian Lihua†

(*School of Software Engineering*, *Xi'an Jiaotong University*, *Xi'an* 710086, *China*)

Abstract: In order to solve the problem of embedding capacity, imperceptibility, real-time and tampering localization in JPEG image watermarking algorithm, this paper proposes a JPEG watermarking algorithm based on entropy coding. Firstly, the algorithm locates the process of JPEG entropy coding to embed the watermarking, which can avoid the forward and reverse quantization operations and ensure that during embedding image changes is small and has good real-time performance. Secondly, according to the principle that some bits in the entropy coding process do not participate in the Huffman coding, the algorithm chooses appropriate coefficients to embed watermark. Our algorithm embeds watermarking information into the least significant bits of these coefficients to avoid the coding error caused by the modification of the Huffman coding coefficient, which should further enhance the imperceptibility of the algorithm. Experiments results show that the algorithm not only has a large embedded capacity and good imperceptibility, but also has high fragility and can accurately locate malicious tampering.

Key words: image watermarking; JPEG image; fragile watermarking; tamper localization

0　引言

随着互联网和多媒体技术的快速发展，JPEG 格式的图像得到了广泛的应用，但是目前对于图像水印算法的研究绝大多数都是针对于 BMP 格式图像，针对 JPEG 压缩域的图像水印算法研究较少。而其中大部分的研究主要集中于 JPEG 压缩域的鲁棒性水印算法[1-3]，用于图像内容的完整性以及真实性验证[4-7]的 JPEG 压缩域脆弱性水印算法的研究相对更少。与鲁棒性水印算法相比，脆弱水印算法在图像受到攻击时会比较敏感[8,9]。文献[10]提出一种可逆的 JPEG 脆弱水印算法，通过折叠量化后系数的哈希值来生成水印，然后将 2 个比特分别嵌入到每个块中 2 个系数的最低有效位中。文献[11]提出基于领域比较的 JPEG 脆弱水印算法，根据 8×8 图像块中的 DCT 系数生成 4 个比特的水印信息，然后根据密钥随机的嵌入到相邻的 4 个图像块 DCT 系数的最低位。上述文献中提出的脆弱水印算法都是在 DCT 系数上嵌入水印，每次水印的嵌入和提取都会涉及到正向和逆向的 DCT 和量化运算，算法的时间复杂度较高。另外虽然上述算法对于常规攻击具有一定的脆弱性，但是对图像质量影响较大，不可感知性较差。文献[12]提出了一种灵活的 JPEG 压缩域水印算法，通过更改 Zig-zag 排序后一维数组中最后一个符合条件的系数的位置来嵌入水印，该算法有一定的脆弱性同时能抵抗特定的噪声攻击，但是该算法不能检测具体的窜改区域。同时由于 Zig-zag 中的顺序对于熵编码过程非常重要，调整系数顺序间接改变最后图像的压缩结果，对图像质量

收稿日期：2018-01-26；**修回日期：**2018-03-30　**基金项目：**国家自然科学基金资助项目（61403302）；中央高校基本科研业务费专项资金资助项目（XJJ2016029）

作者简介：李晨（1981-），女，陕西西安人，讲师，博士，主要研究方向为多媒体技术；喻枭（1995-），男，四川成都人，硕士研究生，主要研究方向为信息安全、图像水印；田丽华（1978-），女（通信作者），陕西西安人，高级工程师，博士，主要研究方向为多媒体技术（lhtian@mail.xjtu.edu.cn）.

的影响较大。文献[13]提出了一种零失真的 JPEG 水印算法，通过同时修改哈夫曼码表和比特流来嵌入水印，由于并没有修改任何 JPEG 量化之后的系数，所以该算法不会对图像造成任何质量影响，虽然该算法脆弱性较好，但是也不能进行窜改定位。

综上，由于熵编码阶段在量化过程之后，避免了正向和逆向的 DCT 和量化运算，能具有很好的透明性和实时性，本文提出了一种基于熵编码的 JPEG 脆弱图像水印算法。算法思想是首先定位 JPEG 熵编码阶段每个块中不会参与哈夫曼编码的比特，然后从中选择交流系数将水印嵌入到其最低有效位中。由于交流系数具有比较敏感的特点，所以本文算法不仅对常见攻击具有良好的脆弱性，而且能够对恶意窜改进行准确定位。

1　JPEG 编码过程及嵌入位置选择

JPEG 图像压缩算法具体流程如图 1 所示，对于 BMP 图像首先进行颜色空间转换，然后对每个颜色分量进行 8×8 分块，接着对每个块进行 DCT 和量化运算，最后经过 Zig-zag 排序和熵编码后以 JPEG 格式保存得到最终的 JPEG 图像。

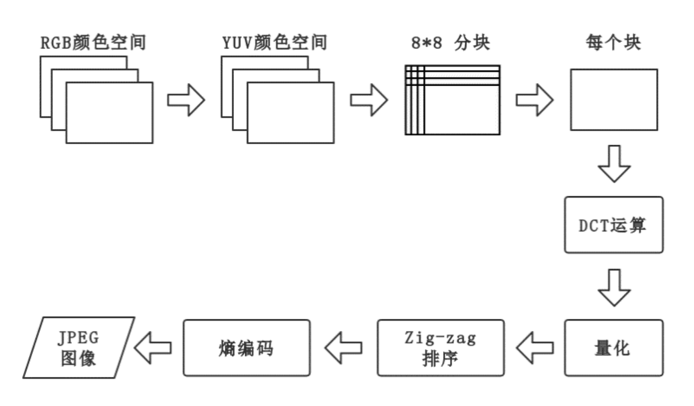


图 1 JPEG 图像编码过程

JPEG 图像压缩算法的熵编码阶段是无损压缩的，具体又包括行程编码、比特编码和哈夫曼编码这三个过程[14]。为了验证嵌入位置的显著性及透明性，必须先介绍 JPEG 图像压缩算法的熵编码过程。在经过前期的分块、DCT、量化以及 Zig-zag 排序过后，实际上熵编码阶段处理的是一个长度为 64 的一维数组，如图 2 所示。

①原始数据	35,7,0,0,0,-6,-2,0,0,-9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,...,0

图 2　熵编码原始数据

可以看到每个 8×8 的分块在经过 DCT 运算尤其是量化之后，矩阵中出现 0 的次数是非常多的。由于一个分块在经过 DCT 变化后，其直流系数会代表其原有分块的大部分能量，一般值都比较大，而右下角剩余的交流系数则代表了原有分块的细节分量，一般值都比较小，且越靠右下角值越小，所以再经过量化后矩阵右下角区域基本都为 0。那么接着将这个分块中量化后的系数按照 Zig-zag 排序后形成的一维数组中，0 的位置会相对比较集中，故熵编码过程的第一步就是对这些 0 进行处理。为了节省编码位数，JPEG 使用了行程编码（RLE）将数组中的非 0 和该数前面的 0 的个数作为一个处理单元。这里需要说明的是每一个处理单元最多只能有 16 个 0，也就是说如果有超过 16 个 0 的情况出现，就需要单独进行分组。如果最后一个单元都是 0，那么就使用一个特殊的标识符 EOB 来表示。行程编码过程如图 3 所示。

①原始数据	35,7,0,0,0,-6,-2,0,0,-9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,...,0							
②RLE编码	35	7	0,0,0,-6	-2	0,0,-9	0,0,...,0,8		0,0,...,0
	35	7	0,0,0,-6	-2	0,0,-9	0,0,...,0	0,0,8	0,0,...,0
	(0,35)	(0,7)	(3,-6)	(0,-2)	(2,-9)	(15,0)	(2,8)	EOB

图 3　行程编码过程

行程编码又具体分为 3 步，第一步是将非 0 和该数前面的 0 划分为一个处理单元，第二步是对每一个单元中有超过 15 个 0 的情况进一步分组，第三步对以上单元再进行编码得到一个二元组。二元组左边的参数代表 0 的个数，右边的参数代表非 0 系数的值。例如（3,-6）代表-6 前面还有 3 个 0，（15,0）表示 0 前面还有 15 个 0，所以（15,0）表示该分组有 16 个 0。

在行程编码结束后，将首先对每个二元组的右边参数进行比特编码，JPEG 对此提供了一张标准的码表[15]。这里比特编码的本质就是将目前的十进制数转换为其二进制编码并记录其二进制码长度，如果参数为正数，那么编码结果就是其二进制编码原码；如果参数是负数，那么就是其正数二进制编码的反码。比如行程编码后得到二元组（3,-6），这时先对-6 进行比特编码，-6 为负数所以其编码结果为其正数的二进制编码的反码，其正数 6 的二进制编码为 110，所以-6 的比特编码为 001，长度是 3，那么比特编码第一步的结果就为（3,3,001）。其中三元组（3,3,001）中参数从左到右分别代表 0 的个数、行程编码后二元组右边参数二进制编码的长度以及其二进制编码。这里需要注意的是，在行程编码中专门说明每一个分组内最多仅有 16 个 0，同时用（15,0）代表连续出现 16 次 0 的情况，所以三元组最左边的参数不会超过 15。并且 JPEG 编码规则中规定，比特编码第一步处理的右边参数只会介于$-2047=-2^{11}$ 到 $2047=2^{11}$ 之间[16]，所以其二进制位数最多为 11 位，也就意味着三元组中间的参数不会超过 11。那么这时可以将三元组的前两个部分合并起来，用一个字节来表示，即高 4 位表示出现 0 的次数，低 4 位表示编码位数。比特编码过程如图 4 所示。

①原始数据	35,7,0,0,0,-6,-2,0,0,-9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,...,0							
②RLE编码	35	7	0,0,0,-6	-2	0,0,-9	0,0,...,0,8		0,0,...,0
	35	7	0,0,0,-6	-2	0,0,-9	0,0,...,0	0,0,8	0,0,...,0
	(0,35)	(0,7)	(3,-6)	(0,-2)	(2,-9)	(15,0)	(2,8)	EOB
③BIT编码	(0,6, ***100011***)	(0,3, ***111***)	(3,3, ***001***)	(0,2, ***01***)	(2,4, ***0110***)	(15,-)	(2,4, ***1000***)	EOB
	(0x6,***100011***)	(0x3,***111***)	(0x33,***001***)	(0x2,***01***)	(0x24,***0110***)	(0xF0,-)	(0x24,***1000***)	EOB

图 4　比特编码过程

接下来还需要进行哈夫曼编码，即还需要对比特编码后得到的二元组的左边参数再进行哈夫曼编码，这也是整个熵编码过程的最后一步。JPEG 提供了两张标准的码表，分别针对于直流系数和交流系数，具体码表可参见文献[16]，整个熵编码过程如图 5 所示。

因为直流分量比交流分量更加重要，将水印嵌入到量化后的直流分量当中会对载体图像的质量产生较大影响[16,17]，所以

本文选择将水印信息嵌入交流系数当中。在图 5 中即从系数 7 开始即可算作交流系数，在比特编码时间其编码值为（0x3,111），从编码过程图 5 可以看出，比特编码右边系数编码值 111 不会参与哈夫曼编码，所以修改右边参数的值不会造成哈夫曼码编码的错位，这样嵌入水印后的图像将具有很好的透明性；同时这种修改也不涉及正向或者逆向的 DCT 和量化操作，所以算法时间具有很好的实时性。并且由于每个分块都会嵌入水印，因此在受到窜改攻击时仅会影响被窜改的块，而未受窜改的块不会受到影响，所以可以依据分块准确定位窜改。综上，本算法选择每个块中交流系数在比特编码时期二元组的最右边参数作为水印的嵌入位置。

①原始数据	35,7,0,0,0,-6,-2,0,0,-9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,...,0													
②RLE编码	35		7		0,0,0,-6		-2		0,0,-9		0,0,...,0,8			0,0,...,0
	35		7		0,0,0,-6		-2		0,0,-9		0,0,...,0	0,0,8		0,0,...,0
	(0,35)		(0,7)		(3,-6)		(0,-2)		(2,-9)		(15,0)	(2,8)		EOB
③BIT编码	(0,6, ***100011***)		(0,3, ***111***)		(3,3, ***001***)		(0,2, ***01***)		(2,4, ***0110***)		(15,-)	(2,4, ***1000***)		EOB
	(0x6,***100011***)		(0x3,***111***)		(0x33,***001***)		(0x2,***01***)		(0x24,***0110***)		0xF0	(0x24,***1000***)		EOB
④哈弗曼编码	***100***	***100011***	***100***	***111***	***1111 1111 0101***	***001***	***01***	***01***	***1111 1111 0100***	***0110***	***1111 1111 001***	***1111 1111 0100***	***1000***	***1010***
⑤序列化	1001000111001111111111010100101011111111101000110111111111001111111111010010001010													
	91 CF FE A5 7F D1 BF CF FA 45													

图 5 熵编码过程

由于本算法是在编码过程嵌入水印，需要在解码过程提取水印，那么即使是脆弱水印算法也必须对编解码过程具有鲁棒性，如果选择最后一个交流系数嵌入水印，该系数过于脆弱，算法在解码部分就无法正确提取水印，不能达到水印提取要求。故本算法选择第一个交流系数嵌入水印，并且在每个块中嵌入 k 次来达到对编解码过程的鲁棒。

2　水印信息的生成

为了进行水印嵌入，首先需要根据水印图片生成可用的水印序列。处理的操作过程为水印信息的二值化、水印置乱处理和水印信息降维，最终的目的是得到一个一维的二值水印数组。具体步骤如下：

a）二值化，就是将水印图像的像素点值转化为 0 或者 255 的过程，实际编码过程中会映射到 0 和 1。首先水印图像为灰度图像，如果是彩色图像需要先转换为灰度图像。本文中映射过程通过阈值来进行，选取作为 128 为阈值，大于等于 128 的像素点值都会被映射为 1，小于 128 的像素点值都映射为 0。

b）Arnold 变换。Arnold 变换是水印算法中使用最广的一种置乱算法，本质是将数字图像中的像素点重新排列，以此来打乱图像的显示。本文使用的 Arnold 置乱具体计算公式如下：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \bmod (N) \qquad (1)$$

其中：（x，y）表示原始图像的像素点，(x'，y')为变换后的像素点，N 为图像的高度。

c）水印降维，即将二维的水印图像信息矩阵转换为一维的数组。可以通过行转换的方式，也可以通过列转换的方式，需要注意的是按哪些方式降维，今后需要同样的方式升维。首先需要计算出一维数组的长度，计算公式如下：

$$C = M \times N \qquad (2)$$

其中：M 代表水印图像的高度，N 代表水印图像的宽度。最终，水印信息可以表示为

$$W_i = \{\mathrm{W}(i),\ 1 \le i \le M \times N\} \qquad (3)$$

3　水印的嵌入和提取算法

3.1　水印嵌入过程

本文基于透明性和实时性要求选择在 DCT 系数量化及 Zig-zag 排序之后的比特流熵编码的过程中嵌入水印。另外如果载体图像是彩色图像，如果同时对 Y, Cr, Cb 通道嵌入水印会对图像质量影响较大。由于 JPEG 一个编码单元中各颜色分量块数的比例为：Y:Cr:Cb=4:1:1，选 Cr, Cb 通道块过少会很可能不满足嵌入容量的要求，故对于彩色图像只选择 Y 分量进行水印嵌入。

为了避免水印在传输过程中可能发生错误的问题，本文需要对能嵌入水印的块进行筛选。从每个块第一个交流系数开始，如果系数非 0 且绝对值大于 1，那么进行标记然后继续往下遍历，直到找到 k 个满足条件的系数后，本文才选择该块嵌入水印。嵌入时将在每一个块中嵌入一个比特的水印信息，每一个水印比特信息将被重复嵌入该块的 k 个符合条件的交流系数中。水印的嵌入过程在编码阶段完成，水印嵌入的流程图如图 6 所示。

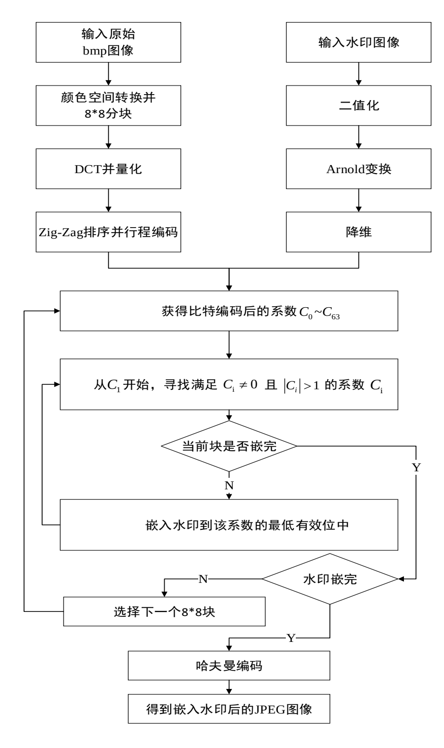


图 6 水印嵌入流程图

具体步骤如下：

a）读取原始的 JPEG 图像，对于每一个 8×8 的块，在熵编码阶段，执行哈夫曼解码和行程解码以获得比特编码时期的最右系数的值，其值记为 $Z_i(j)$，其中 $0 \le j \le 63$, i=1,2,. . . , N 。

b）对于每一个 8×8 的块，则需要找到从第一个交流系数开始，满足非 0 且绝对值大于 1 的系数，定位其在二进制码流中的位置，具体定义如下所示：

$$\mathrm{Pos}_i = Min(j) : Z_i(j) \ne 0 \quad | Z_i(j) | > 1,\ 1 \le j \le 63,\ i=1,2,...,N \tag{4}$$

此时，第一个满足条件的系数（First Appropriate Data，FAD）即为

$$FAD = Z_i(Pos_i) \tag{5}$$

c）从 FAD 的位置开始依次往下遍历，如果还有系数满足非 0 且绝对值大于 1 的条件，那么进行标记然后继续往下遍历，直到找到 k 个满足条件的系数后，跳转到下一个步骤；如果在该块中没有找到 k 个满足条件的系数，则跳过这个块，对下一个块从步骤 2 开始继续进行遍历选择。

d）对于每个选定的 8×8 块，根据水印信息 W_i 的值来依次修改 k 个系数的值，具体每个系数的修改过程如下所示：

$$C_{u,v}^{AC}{}' = \begin{cases} C_{u,v}^{AC}, & C_{u,v}^{AC} \text{ 为偶数并且 } W_i = 0 \\ C_{u,v}^{AC}+1, & C_{u,v}^{AC} \text{ 为偶数并且 } W_i = 1 \\ C_{u,v}^{AC}, & C_{u,v}^{AC} \text{ 为奇数并且 } W_i = 1 \\ C_{u,v}^{AC}-1, & C_{u,v}^{AC} \text{ 为奇数并且 } W_i = 0 \end{cases} \tag{6}$$

其中：$C_{u,v}^{AC}$ 代表当前需要嵌入水印的原系数值，u, v 下标代表其在矩阵块中的位置，AC 代表其为交流系数。$C_{u,v}^{AC}{}'$ 为对应第 i 个系数嵌入水印比特后的新系数值，W_i 为第 i 个块对应的一维水印的比特值。

e）对于每个符合条件的 8×8 块，在码流中用得到的新的值 $C_{u,v}^{AC}{}'$ 依次替换原始的 $C_{u,v}^{AC}$，然后再进行哈夫曼编码，从而得到了最终的嵌入水印信息的 JPEG 图像序列。

3.2　水印提取过程

在水印提取阶段，需要在 JPEG 的解码过程中提取水印。同样，对于每个 8×8 的图像块，都需要进行哈夫曼解码，然后首先定位到每一个符合条件的块中的第一个非 0 且绝对值大于 1 的交流系数（First Appropriate Data，FAD），根据 FAD 的位置依次往后遍历，按序找到符合条件的 k 个系数，然后从每一个系数中提取一个比特的水印信息，统计从 k 个系数中提取出的水印信息的值，根据 1 出现的次数来决定当前块中提取的水印信息的值。将所有块中的水印信息提取后，再进行一维水印数组的升维、反置乱处理、以图片格式保存，即得到了提取出的水印图像。水印提取的流程图如图 7 所示，具体步骤如下：

a）读取嵌入水印后的 JPEG 图像，对于每一个 8×8 的块，在熵编码阶段，执行哈夫曼解码到比特编码以获得比特编码时期的最右系数的值，其值记为：$Z_i'(j)$，其中：$0 \le j \le 63$, i=1,2,. . . , N 。

b）定位到 $Z_i'(j)$ 中的第一个非 0 且绝对值大于 1 的交流系数，即 FAD'。

c）从 FAD' 的位置开始依次往下遍历，如果系数 $C_{u,v}^{AC}{}'$ 满足非 0 且绝对值大于 1 的条件，那么进行标记然后继续往下遍历，直到找到 k 个满足条件的系数后，则选中该块进行水印提取，进行下一个步骤；如果在该块中没有找到 k 个满足条件的系数，就跳过这个块，对下一个块从步骤 2 开始继续进行遍历选择。

d）在被选中的块中，根据系数 $C_{u,v}^{AC}{}'$ 的值来从每个系数提取一个比特的水印信息，统计 k 个系数中提出的 1 的次数，具体公式如下：

$$Count = \begin{cases} Count & C_{u,v}^{AC}{}' \text{ 为偶数} \\ Count+1 & C_{u,v}^{AC}{}' \text{ 为奇数} \end{cases} \tag{7}$$

其中：Count 初始值为 0，如果当前系数值为偶数，Count 值不变；如果当前系数值为奇数，Count 值自增 1。即统计 k 个系数中 1 出现的次数，然后根据当前块中 1 出现的次数获得当前块中提取的水印信息。具体公式如下：

$$W_i' = \begin{cases} 0, & Count < \dfrac{k}{2} \\ 1, & Count \ge \dfrac{k}{2} \end{cases} \tag{8}$$

其中：Count 代表该块中从 k 个系数中提取的信息中 1 出现的次数。W_i' 表示从符合条件的块中提出的第 i 个比特的水印信息。

e）对提取出的一维水印数组进行升维、Arnold 逆变换、以图像格式输出，得到提取出的水印图像。

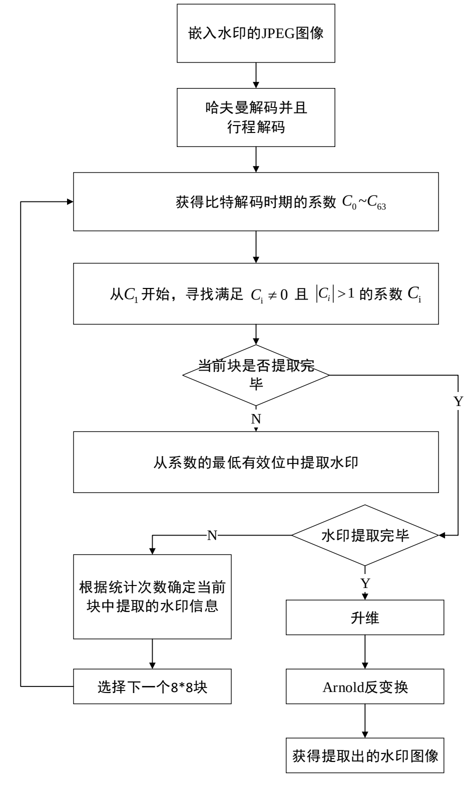


图 7　水印提取流程图

4　实验结果及分析

4.1　实验环境

本文测试环境为：Windows10 旗舰版操作系统，Intel Core i5 处理器，8GB 内存。所有测试均在 Visual Studio 2012 下运行，测试过程中为了便于与参考文献使用的载体图像进行对比，使用了水印算法中经典的载体图像，分别为 Lena、Pepper、Airplane、Boat，大小均为 512×512，量化因子为 100。水印图像大小为 64×64，原始水印图像如图 8 所示。经过多次实验测试，*k* 的值取 5 时效果达到最佳，实验中 *k* 取值为 5。



图 8　原始水印图像

4.2　不可感知性和嵌入容量测试

不可感知性，又被称作透明性，使用来衡量图像水印算法对原始图像载体可见性的影响程度，不可感知性一般通过两个方面来进行评价，分别是主观和客观。主观性评价是在载体图像和嵌入容量都一致的情况下，嵌入水印后的图像载体对人眼的视觉改变量越小代表算法的透明性越好。客观性评价一般通过计算峰值信噪比（PSNR）来衡量，在载体图像和嵌入容量都一致的情况下，原始载体图像和嵌入水印后的载体图像的 PSNR 最高代表水印算法的透明性越好。首先，从主观性上进行评价，如图 9 和 10 所示，原始载体图像和嵌入水印后的载体图像，在人眼视觉上面并无任何的区别，没有任何嵌入过水印图像的痕迹，说明算法的透明性良好。其次，从 PSNR 的结果上看，本算法对于 Lena 图像，其 PSNR 为 59.6531；Pepper 图像的 PSNR 为 59.9543；Airplane 图像的 PSNR 为 59.5651；Boat 图像的 PSNR 为 56.3443，可以看出 4 张测试图像的 PSNR 都在 55 dB 以上。而同类的其他算法，PSNR 基本在 45 dB 左右，可见在不可感知性方面本文算法有很大的优势。

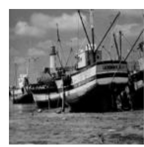
   

图 9　原始载体图像

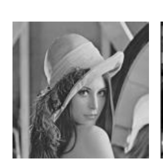
   

图 10　嵌入水印后的载体图像

其次，在水印的嵌入容量上，由于本文是在每个 8×8 块中选择 5 个系数来嵌入水印，所以每个块中会嵌入 5 个比特。由此计算，本文算法的水印容量为 0.078125bpp（bit per pixel）。最后，为了更好的展现本算法的优势，本文与文献[10，11]进行了对比，对比结果如表 1 所示。

表 1　本文算法与文献算法嵌入容量和透明性对比

载体图像 512×512	本文算法 嵌入容量/ 10^3 bpp	PSNR	文献[10]算法 嵌入容量/ 10^3 bpp	PSNR	文献[11]算法 嵌入容量/ 10^3 bpp	PSNR
Lena	78.13	59.65	31.25	45.25	62.5	43.45
Pepper	78.13	59.95	31.25	46.02	62.5	44.25
Airplane	78.13	59.57	31.25	45.76	62.5	43.52
Boat	78.13	56.34	31.25	44.93	62.5	45.06

由表 1 可以看出，本文算法在嵌入容量和 PSNR 上均远高于文献[10，11]的算法。在文献[10]中，选择在每个量化后的块中嵌入 2 个比特的水印信息，由于直接修改量化系数后在哈夫曼编码时造成了误差，因此 PSNR 远低于本文算法；文献[11]算法利用每个块的相关性来生成水印信息，在每个块中嵌入 4 个比特，虽然文献[11]在嵌入容量上与本文差距不大，但是 PSNR 依然远低于本文算法。另外，相比对比算法来说，本文算法在水印嵌入和水印提取更加简单；同时由于不需要经过反量化和反 DCT 运算即可提取水印，非常适合实时应用。提取水印运行时间对比如表 2 所示：

表 2　提取水印运行时间对比

载体图像大小	本文算法 运行时间/s	文献[10]算法 运行时间/s	文献[11]算法 运行时间/s
256×256	0.608	1.806	1.986
512×512	1.157	5.739	5.556
1024×1024	1.155	13.716	16.894
2048×2048	1.209	36.377	42.045

其中，大小为 256×256 载体图像中嵌入的是 32×32 大小水印图像，其余载体图像都是嵌入 64×64 大小水印图像。本文算法将水印嵌入到熵编码过程中，提取水印时只需要进行部分解码，并且一旦水印提取完毕即可停止解码过程，时间复杂度与水印大小成正相关关系。而对比文献算法需要进行完全解码后才能提取水印，解码流程复杂，时间复杂度高。

4.3　脆弱性测试

对于脆弱水印要求其对于一些常见的处理操作比较敏感，通过提取水印信息与重构水印的 BER 和 NC 值对算法的脆弱性进行评价。为测试本文所提出方案的脆弱性，对嵌入水印后的载体图像分别做了如下几种攻击操作：

a）添加方差为 0.002 的高斯噪声；

b）添加方差为 0.002 的椒盐噪声；

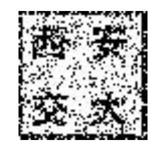
c）使用 3×3 模板的低通滤波；

d）使用 3×3 模板的中值滤波；

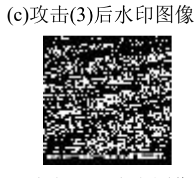
e）重压缩，量化因子为 90；

f）重压缩，量化因子为 80；

图 11 给出了本文算法受到以上攻击之后的提取出的水印图像。从图中可以看出本文算法受到攻击后提取出的水印图像均被破坏，对常规攻击都比较敏感。


(a)攻击(1)后的水印图像 (b)攻击(2)后水印图像 (c)攻击(3)后水印图像



(d)攻击(4)后的水印图像 (e)攻击(5)后水印图像 (f)攻击(6)后水印图像

图 11 受到攻击后提取出的水印图像

表 3 给出了经过上述攻击后的载体图像提取出的水印图像与原始水印图像的 NC 和 BER 值。从表 3 可以看出在大部分攻击下，本文算法 NC 比较小，BER 比较大，也就是说提取出的水印和原始的水印差异较大，可见本文算法的脆弱性较好。此外表 3 还给出了本文算法与文献[10]和文献[11]算法的对比结果，通过对比发现本文水印方案在受到所列攻击时，绝大部分攻击操作下的 NC 均小于文献[10,11]，BER 均大于文献[10,11]，说明本文所提的方案的脆弱性要优于对比文献。

表 3 本文算法和文献算法脆弱性对比

攻击类型	本文算法		文献[10]算法		文献[11]算法	
	BER	NC	BER	NC	BER	NC
无攻击	0	1	0	1	0	1
高斯噪声	0.5090	0.5853	0.2021	0.7905	0.4823	0.5923
椒盐噪声	0.1609	0.8811	0.4406	0.3236	0.4175	0.4750
低通滤波	0.5012	0.5886	0.4266	0.5818	0.4703	0.6270
中值滤波	0.5154	0.5702	0.5083	0.5969	0.4056	0.5855
重量化 90	0.5471	0.5172	0.1064	0.8896	0.1186	0.8513
重量化 80	0.5425	0.4987	0.2021	0.7886	0.1895	0.9432

4.4 窜改定位测试

本文算法是基于哈夫曼解码和行程解码后，获取比特编码时期二元组中右边的参数来进行水印提取，如果载体图像的内容没有发生恶意的窜改，那么经过解码后获得的相应参数值不会改变。如果载体图像被恶意窜改，那么窜改区域图像块中相应获得的参数会大幅的改变，从而从该窜改区域提取出的水印也肯定不同，可以据此进行窜改定位。

使用数组 $w_1(i)$ 表示原始水印图像生成的一维水印数组，$w_2(i)$ 表示从嵌入水印的载体图像中提取出的一维水印数组，$m(i)$ 为原始水印图像生成的一维水印数组和提取出的一维水印数组的差值并取绝对值的结果，如下所示：

$$m(i) = |w_1(i) - w_2(i)| \tag{9}$$

其中：i 代表图像中的第 i 块，$m(i)$ 的值为 0，那么说明两个块的变化不大，相反则认为该图像块已经发生改变，通过 $m(i)$ 即可以定位到不同块的位置。

为了测试本文算法针对 JPEG 图像被恶意窜改时的检测和定位方面的性能，本文进行了如下的实验操作。本文使用的窜改图像均为 Lena，分别进行了 3 次窜改测试，第一次测试如图 12(a)所示，将 Lena 图左下角用 Lena 的头发窜改替代；第二次测试如图 12(b)所示，窜改 Lena 的帽子位置；第三次测试如图 12(c)所示，分别窜改 Lena 的左上角和右下角的两个位置。经过窜改的载体图像相应的位置定位如图 13 所示，测试表明本文算法能够实现对窜改攻击的图像的定位。





(a)图像左下角被窜改 (b)图像帽子左上被窜改 (c)图像左上右下被窜改

图 12 窜改图像

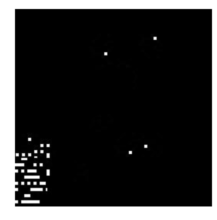
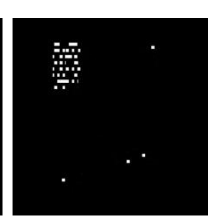
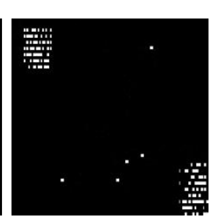




图 13 窜改定位检测结果

5 结束语

本文利用熵编码阶段某些比特不会参与哈夫曼编码这一特性，提出了一种基于熵编码的 JPEG 压缩域脆弱水印算法。本算法选择将水印信息冗余嵌入到每个图像块的 k 个 DCT 交流系数熵编码后的最低有效位中，一方面可以提高算法的稳定性，另一方面还可避免量化损失和哈夫曼编码误差等问题，使得算法透明性较高。实验结果表明，算法具有较大的嵌入容量和很好的不可感知性，对常规攻击脆弱性较高且能够对恶意窜改进行准确定位。算法实现过程简单，仅需对编解器进行较少改动即可实现，在实现水印的盲提取的同时保证了算法的效率，提高了算法的实用性。

参考文献：

[1] Caragata D, Radu A L, El Assad S. Chaos based fragile watermarking algorithm for JPEG images [C]// Proc of International Conference on Internet Technology and Secured Transactions. 2010: 1-7.

[2] 常玉红，黄惠芬，王志红. 检测图像窜改的脆弱水印技术 [J]. 网络与信息安全学报, 2017, 3 (07): 47-52. (Chang Yuhong, Huang Huifen, Wang Zhihong. Fragile watermarking technique for detecting image [J]. Chinese Journal of Network and Information Security, 2017, 3 (7): 47-52.)

[3] Chan Haotang, Hwang Wenjyihwang, Cheng Chaujern. Digital hologram authentication using a hadamard-based reversible fragile watermarking algorithm [J]. Journal of Display Technology, 2015, 11 (2): 193-203.

[4] Munir R. A chaos-based fragile watermarking method in spatial domain for image authentication [C]// Proc of International Seminar on Intelligent Technology and ITS Applications. 2015: 227-232.

[5] Üstübioğlu A, Ulutaş G, Üstübioğlu B. Tamper localization of the medical images based on fragile watermarking [C]// Proc of Signal Processing and Communications Applications Conference, 2017: 1 - 4.

[6] Singh A, Dutta M K. A blind & fragile watermarking scheme for tamper detection of medical images preserving ROI [C]// Proc of International Conference on Medical Imaging, M-Health and Emerging Communication Systems. 2015: 230-234.

[7] Cheng C J, Hwang W J, Zeng Hanyi. A fragile watermarking algorithm for hologram authentication [J]. Journal of Display Technology, 2014, 10 (4): 263-271.

[8] 樊洁，李建军. 一种半脆弱数字水印算法 [J]. 计算机技术与发展, 2017, 27 (02): 69-71. (Fan Jie, Li Jianjun. A semi-fragile digital watermarking algorithm [J]. Computer Technology and Development, 2017, 27 (2): 69-71.)

[9] Dong Shusen, Li Jianfei, Liu Shouxun. Frequency domain digital watermark algorithm implemented in spatial domain based on correlation coefficient and quadratic DCT transform [C]// Proc of IEEE International Conference on Progress in Informatics and Computing. 2017: 596-600.

[10] Zhang Xinpeng, Wang Shuozhong, Qian Zhenxing. Reversible fragile watermarking for locating tampered blocks in JPEG images [J]. Multimedia Tools & Applications, 2010, 90 (12): 3026-3036.

[11] 霍耀冉，和红杰，陈帆. 基于邻域比较的 JPEG 脆弱水印算法及性能分析 [J]. 软件学报, 2012, 23 (09): 2510-2521. (Huo Yaoran, He Hongjie, Chen Fan. Fragile watermarking algorithm for JPEG images based on neighborhood comparison and its performance analysis [J]. Journal of Software, 2012, 23 (09): 2510-2521.)

[12] Fallahpour Mehdi, Megías David. Flexible image watermarking in JPEG domain [C]// Proc of IEEE International Symposium on Signal Processing and Information Technology. 2017: 311-316.

[13] Wu Yongdong, Deng R H. Zero-error watermarking on jpeg images by shuffling huffman tree nodes [C]// Visual Communications and Image Processing. 2011: 1-4.

[14] 宫泽林. 基于 JPEG 图像压缩及其仿真实现 [J]. 中国科技信息, 2013 (13): 84-84. (Gong Zelin. Image compression and simulation implementation based on JPEG [J]. China Science and Technology Information, 2013 (13): 84-84.)

[15] 杨雨薇，张亚萍，李幸刚. 一种改进的 JPEG 图像压缩编码算法 [J]. 云南师范大学学报：自然科学版, 2016, 36 (6): 32-39. (Yang Yuwei, Zhang Yaping, Li Xinggang. An improved JPEG image compression coding algorithm [J]. Journal of Yunnan Normal University: Natural Sciences Edition, 2016, 36 (6): 32-39.)

[16] Schaefer G. Fast JPEG compressed domain image retrieval [C]// Proc of International Conference on Vision, Image and Signal Processing. 2017: 22-26.

[17] Dong Hualin, He Mingyuan, Qiu Ming. Optimized gray-scale image watermarking algorithm based on DWT-DCT-SVD and chaotic firefly algorithm [C]// Proc of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. 2015: 311-313.

[18] Vo P H, Nguyen T S, Huynh V T, *et al.* A robust hybrid watermarking scheme based on DCT and SVD for copyright protection of stereo images [C]// Proc of the 4th Nafosted Conference on Information and Computer Science. 2017: 331-335.